université de BORDEAUX

Formation CNRS
18 Novembre 2016
Python pour la biologie

Biopython: Multiple alignments

cgfb
BIOINFORMATIQUE

université de BORDEAUX

Introduction

- A collection of multiple sequences which have been aligned together
- Insertion of gap characters, and addition of leading or trailing gaps
  - such that all the sequence strings are the same length.
- Can be regarded as a matrix of letters, where each row is held as a SeqRecord object internally
- Introduce :
  - MultipleSeqAlignment object which holds this kind of data
  - Bio.AlignIO module for reading and writing them as various file formats (following the design of the Bio.SeqIO module)

Parsing or Reading Sequence Alignments

- Bio.AlignIO.read() and Bio.AlignIO.parse()
- Using Bio.AlignIO.parse() will return an iterator which gives MultipleSeqAlignment objects
- Bio.AlignIO.read() function which returns a single MultipleSeqAlignment object
- First argument is a handle to read the data from, typically an open file, or a filename
- Second argument is a lower case string specifying the alignment format. (see http://biopython.org/wiki/AlignIO)
- Optional seq_count argument for dealing with ambiguous file formats
- Optional Alphabet argument allowing you to specify the expected alphabet.

Single Alignments

➔ consider the following annotation rich protein alignment in the PFAM or Stockholm file

```
# STOCKHOLM 1.0
#=GS COATB_BPIKE/30-81 AC P03620.1
#=GS COATB_BPIKE/30-81 DR PDB; 1ifl ; 1-52;
#=GS Q9T0Q8_BPIKE/1-52 AC Q9T0Q8.1
#=GS COATB_BPI22/32-83 AC P15416.1
#=GS COATB_BPM13/24-72 AC P69541.1
#=GS COATB_BPM13/24-72 DR PDB; 2cpb ; 1-49;
#=GS COATB_BPM13/24-72 DR PDB; 2cps ; 1-49;
#=GS COATB_BPZJ2/1-49 AC P03618.1
#=GS Q9T0Q9_BPFD/1-49 AC Q9T0Q9.1
#=GS Q9T0Q9_BPFD/1-49 DR PDB; 1nh4 A; 1-49;
#=GS COATB_BPIF1/22-73 AC P03619.2
#=GS COATB_BPIF1/22-73 DR PDB; 1ifk ; 1-50;
COATB_BPIKE/30-81 AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA
#=GR COATB_BPIKE/30-81 SS -HHHHHHHHHHHHHH--HHHHHHHH--HHHHHHHHHHHHHHHHHHHHH----
Q9T0Q8_BPIKE/1-52 AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA
COATB_BPI22/32-83 DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA
#=GR COATB_BPM13/24-72 SS ---S-T...CHCHHHHCCCCTCCCTTCHHHHHHHHHHHHHHHHHHHHCTT--
COATB_BPZJ2/1-49 AEGDDP...AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA
Q9T0Q9_BPFD/1-49 AEGDDP...AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
#=GR Q9T0Q9_BPFD/1-49 SS ------...-HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH--
COATB_BPIF1/22-73 FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA
#=GR COATB_BPIF1/22-73 SS XX-HHHH--HHHHHH--HHHHHHH--HHHHHHHHHHHHHHHHHHHHHH---
#=GC SS_cons XHHHHHHHHHHHHHHHCHHHHHHHHCHHHHHHHHHHHHHHHHHHHHHC--
#=GC seq_cons AEssss...AptAhDSLpspAT-hIu.sWshVsslVsAsluIKLFKKFsSK
```

Single Alignments (2)

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
```

```
>>> print(alignment)
SingleLetterAlphabet() alignment with 7 rows and 52 columns
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRL...SKA COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKL...SRA Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRL...SKA COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKL...SRA COATB_BPIF1/22-73
```

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
>>> print("Alignment length %i" % alignment.get_alignment_length())
Alignment length 52
>>> for record in alignment:
... print("%s - %s" % (record.seq, record.id))
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA - COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA - Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA - COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA - COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA - COATB_BPIF1/22-73
```

Single Alignments (3)

- Include database cross-references to the PDB and the associate known secondary structure

```
>>> for record in alignment:
... if record.dbxrefs:
... print("%s %s" % (record.id, record.dbxrefs))
COATB_BPIKE/30-81 ['PDB; 1ifl ; 1-52;']
COATB_BPM13/24-72 ['PDB; 2cpb ; 1-49;', 'PDB; 2cps ; 1-49;']
Q9T0Q9_BPFD/1-49 ['PDB; 1nh4 A; 1-49;']
COATB_BPIF1/22-73 ['PDB; 1ifk ; 1-50;']
```

```
>>> for record in alignment:
... print(record)
>COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA
>Q9T0Q8_BPIKE/1-52
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA
>COATB_BPI22/32-83
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA
>COATB_BPM13/24-72
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
>COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA
>Q9T0Q9_BPFD/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
>COATB_BPIF1/22-73
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA
```

Single Alignments (4)

➔ download and save this as file “PF05371 seed.fa”

```
from Bio import AlignIO
alignment = AlignIO.read("PF05371_seed.faa", "fasta")
print(alignment)
```

➔ no annotation nor database cross-references because these are not included in the FASTA file format

Multiple alignments (1)

- Suppose you have a small alignment in PHYLIP format

```
5 6
Alpha AACAAC
Beta AACCCC
Gamma ACCAAC
Delta CCACCA
Epsilon CCAAAC
```

- To bootstrap a phylogenetic tree using the PHYLIP tools:
  - create a set of many resampled alignments using the tool bootseq
- This would give output (abbreviated for conciseness):

```
5 6
Alpha AAACCA
Beta AAACCC
Gamma ACCCCA
Delta CCCAAC
Epsilon CCCAAA
5 6
Alpha AAACAA
Beta AAACCC
Gamma ACCCAA
Delta CCCACC
Epsilon CCCAAA
5 6
Alpha AAAAAC
Beta AAACCC
Gamma AACAAC
Delta CCCCCA
```

Multiple alignments (2)

➔ If you wanted to read this in using Bio.AlignIO you could use

```
from Bio import AlignIO
alignments = AlignIO.parse("resampled.phy", "phylip")
for alignment in alignments:
print(alignment)
print("")
```

➔ abbreviated for display:

```
SingleLetterAlphabet() alignment with 5 rows and 6 columns
AAACCA Alpha
AAACCC Beta
ACCCCA Gamma
CCCAAC Delta
CCCAAA Epsilon
SingleLetterAlphabet() alignment with 5 rows and 6 columns
AAACAA Alpha
AAACCC Beta
ACCCAA Gamma
CCCACC Delta
CCCAAA Epsilon
...
SingleLetterAlphabet() alignment with 5 rows and 6 columns
AAAACC Alpha
ACCCCC Beta
AAAACC Gamma
CCCCAA Delta
CAAACC Epsilon
```

Multiple alignments (3)

- As with the function Bio.SeqIO.parse() , using Bio.AlignIO.parse() returns an iterator.
- to keep all the alignments in memory at once
- Turn the iterator into a list

```
from Bio import AlignIO
alignments = list(AlignIO.parse("resampled.phy", "phylip"))
last_align = alignments[-1]
first_align = alignments[0]
```

Ambiguous alignments (&)

- Many alignment file formats can explicitly store more than one alignment,

```
>Alpha
ACTACGACTAGCTCAG--G
>Beta
ACTACCGCTAGCTCAGAAG
>Gamma
ACTACGGCTAGCACAGAAG
>Alpha
ACTACGACTAGCTCAGG--
>Beta
ACTACCGCTAGCTCAGAAG
>Gamma
ACTACGGCTAGCACAGAAG
```

- What about this next example?

```
>Alpha
ACTACGACTAGCTCAG--G
>Beta
ACTACCGCTAGCTCAGAAG
>Alpha
ACTACGACTAGCTCAGG--
>Gamma
ACTACGGCTAGCACAGAAG
>Alpha
ACTACGACTAGCTCAGG--
>Delta
ACTACGGCTAGCACAGAAG
```

- To interpret these FASTA examples as several separate alignments, we can use Bio.AlignIO.parse() with the optional seq_count argument which specifies how many sequences are expected in each alignment (3,2,2 in the previous example

```
for alignment in AlignIO.parse(handle, "fasta", seq_count=2):
print("Alignment length %i" % alignment.get_alignment_length())
for record in alignment:
print("%s - %s" % (record.seq, record.id))
print("")
```

```
Alignment length 19
ACTACGACTAGCTCAG--G - Alpha
ACTACCGCTAGCTCAGAAG - XXX
Alignment length 17
ACTACGACTAGCTCAGG - Alpha
ACTACGGCAAGCACAGG - YYY
Alignment length 21
--ACTACGAC--TAGCTCAGG - Alpha
GGACTACGACAATAGCTCAGG - ZZZ
```

Writing alignments (1)

- Bio.AlignIO.write() taking three arguments:
  - Some MultipleSeqAlignment objects (or for backwards compatibility the Obsolete Alignment objects),
  - a handle or filename to write to
  - a sequence format

```
from Bio.Alphabet import generic_dna
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.Align import MultipleSeqAlignment
align1 = MultipleSeqAlignment([
SeqRecord(Seq("ACTGCTAGCTAG", generic_dna), id="Alpha"),
SeqRecord(Seq("ACT-CTAGCTAG", generic_dna), id="Beta"),
SeqRecord(Seq("ACTGCTAGDTAG", generic_dna), id="Gamma"),
])
align2 = MultipleSeqAlignment([
SeqRecord(Seq("GTCAGC-AG", generic_dna), id="Delta"),
SeqRecord(Seq("GACAGCTAG", generic_dna), id="Epsilon"),
SeqRecord(Seq("GTCAGCTAG", generic_dna), id="Zeta"),
])
align3 = MultipleSeqAlignment([
SeqRecord(Seq("ACTAGTACAGCTG", generic_dna), id="Eta"),
SeqRecord(Seq("ACTAGTACAGCT-", generic_dna), id="Theta"),
SeqRecord(Seq("-CTACTACAGGTG", generic_dna), id="Iota"),
])
my_alignments = [align1, align2, align3]
```

➔ Now we have a list of Alignment objects, we'll write them to a PHYLIP format file:

```
from Bio import AlignIO
AlignIO.write(my_alignments, "my_example.phy", "phylip")
```

➔ open this file in your favourite text editor it should look like this:

```
3 12
Alpha ACTGCTAGCT AG
Beta ACT-CTAGCT AG
Gamma ACTGCTAGDT AG
3 9
Delta GTCAGC-AG
Epislon GACAGCTAG
Zeta GTCAGCTAG
3 13
Eta ACTAGTACAG CTG
Theta ACTAGTACAG CT-
Iota -CTACTACAG GTG
```

Converting between sequence alignment file formats (1)

- For this example, we'll load the PFAM/Stockholm format file used earlier and save it as a Clustal W format file:

```
from Bio import AlignIO
count = AlignIO.convert("PF05371_seed.sth", "stockholm", "PF05371_seed.aln", "clustal")
print("Converted %i alignments" % count)
```

- we gave it the alignment iterator returned by Bio.AlignIO.parse()

```
from Bio import AlignIO
alignments = AlignIO.parse("PF05371_seed.sth", "stockholm")
count = AlignIO.write(alignments, "PF05371_seed.aln", "clustal")
print("Converted %i alignments" % count)
```

- Or, using Bio.AlignIO.parse() and Bio.AlignIO.write()

```
from Bio import AlignIO
alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
AlignIO.write([alignment], "PF05371_seed.aln", "clustal")
```

Converting between sequence alignment file formats (2)

- you should end up with the same new Clustal W format file “PF05371 seed.aln" with the following content

```
CLUSTAL X (1.81) multiple sequence alignment
COATB_BPIKE/30-81 AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSS
Q9T0Q8_BPIKE/1-52 AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVS
COATB_BPI22/32-83 DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSS
COATB_BPM13/24-72 AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTS
COATB_BPZJ2/1-49 AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFAS
Q9T0Q9_BPFD/1-49 AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTS
COATB_BPIF1/22-73 FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVS
COATB_BPIKE/30-81 KA
Q9T0Q8_BPIKE/1-52 RA
COATB_BPI22/32-83 KA
COATB_BPM13/24-72 KA
COATB_BPZJ2/1-49 KA
Q9T0Q9_BPFD/1-49 KA
COATB_BPIF1/22-73 RA
```

- you could make a PHYLIP format le which we'll name “PF05371 seed.phy"

```
from Bio import AlignIO
AlignIO.convert("PF05371_seed.sth", "stockholm", "PF05371_seed.phy", "phylip")
```

Converting between sequence alignment file formats (3)

```
7 52
COATB_BPIK AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIRLFKKFSS
Q9T0Q8_BPI AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIKLFKKFVS
COATB_BPI2 DGTSTATSYA TEAMNSLKTQ ATDLIDQTWP VVTSVAVAGL AIRLFKKFSS
COATB_BPM1 AEGDDP---A KAAFNSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
COATB_BPZJ AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFAS
Q9T0Q9_BPF AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
COATB_BPIF FAADDATSQA KAAFDSLTAQ ATEMSGYAWA LVVLVVGATV GIKLFKKFVS
KA
RA
.....
```

➔ Big handicaps of the original PHYLIP alignment file format is that the sequence identifiers are strictly truncated at ten characters.

```
from Bio import AlignIO
AlignIO.convert("PF05371_seed.sth", "stockholm", "PF05371_seed.phy", "phylip-relaxed")
7 52
COATB_BPIKE/30-81 AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIRLFKKFSS
Q9T0Q8_BPIKE/1-52 AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIKLFKKFVS
COATB_BPI22/32-83 DGTSTATSYA TEAMNSLKTQ ATDLIDQTWP VVTSVAVAGL AIRLFKKFSS
COATB_BPM13/24-72 AEGDDP---A KAAFNSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
COATB_BPZJ2/1-49 AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFAS
Q9T0Q9_BPFD/1-49 AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
COATB_BPIF1/22-73 FAADDATSQA KAAFDSLTAQ ATEMSGYAWA LVVLVVGATV GIKLFKKFVS
KA
RA
....
```

Converting between sequence alignment file formats (4)

- have to work with the original strict PHYLIP format, then you may need to compress the identifiers somehow or assign your own names or numbering system.

```
from Bio import AlignIO
alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
name_mapping = {}
for i, record in enumerate(alignment):
name_mapping[i] = record.id
record.id = "seq%i" % i
print(name_mapping)
AlignIO.write([alignment], "PF05371_seed.phy", "phylip")
```

- use a Python dictionary to record a simple mapping from the new sequence system to the original identifier:

```
{0: 'COATB_BPIKE/30-81', 1: 'Q9T0Q8_BPIKE/1-52', 2: 'COATB_BPI22/32-83', ...}
```

```
7 52
seq0 AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIRLFKKFSS
seq1 AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIKLFKKFVS
seq2 DGTSTATSYA TEAMNSLKTQ ATDLIDQTWP VVTSVAVAGL AIRLFKKFSS
seq3 AEGDDP---A KAAFNSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
seq4 AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFAS
seq5 AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
seq6 FAADDATSQA KAAFDSLTAQ ATEMSGYAWA LVVLVVGATV GIKLFKKFVS
KA
RA
```

Getting your alignment in formatted string (1)

- Take advantage of the alignment object's format() method

```
from Bio import AlignIO
alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
print(alignment.format("clustal"))
```

- The SeqRecord object has a similar method using output formats supported by Bio.SeqIO
- The format() method is using the StringIO string based handle and calling Bio.AlignIO.write()

```
from Bio import AlignIO
from StringIO import StringIO
alignments = AlignIO.parse("PF05371_seed.sth", "stockholm")
out_handle = StringIO()
AlignIO.write(alignments, out_handle, "clustal")
clustal_data = out_handle.getvalue()
print(clustal_data)
```

Slicing alignments (1)

- Alignment objects act like a Python list of SeqRecord objects (the rows)

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
>>> print("Number of rows: %i" % len(alignment))
Number of rows: 7
>>> for record in alignment:
... print("%s - %s" % (record.seq, record.id))
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA - Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA - COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA - COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA - COATB_BPIF1/22-73
```

- Suppose you have a small alignment in PHYLIP format

```
>>> print(alignment)
SingleLetterAlphabet() alignment with 7 rows and 52 columns
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKL...SRA Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRL...SKA COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKL...SRA COATB_BPIF1/22-73
>>> print(alignment[3:7])
SingleLetterAlphabet() alignment with 4 rows and 52 columns
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKL...SRA COATB_BPIF1/22-73
```

Alignments as array (1)

➔ If you wanted to select by column?

```
>>> print(alignment[2, 6])
T
```

➔ Using two integer indices pulls out a single letter

```
>>> print(alignment[2].seq[6])
T
```

➔ pull out a single column as a string like this

```
>>> print(alignment[:, 6])
TTT---T
```

➔ Select a range of columns

```
>>> print(alignment[3:6, :6])
SingleLetterAlphabet() alignment with 3 rows and 6 columns
AEGDDP COATB_BPM13/24-72
AEGDDP COATB_BPZJ2/1-49
AEGDDP Q9T0Q9_BPFD/1-49
```

➔ Leaving the first index as means take all the rows:

```
>>> print(alignment[:, :6])
SingleLetterAlphabet() alignment with 7 rows and 6 columns
AEPNAA COATB_BPIKE/30-81
AEPNAA Q9T0Q8_BPIKE/1-52
.....
```

Slicing alignments (1)

➔ columns 7, 8 and 9 which are gaps in three of the seven sequences

```
>>> print(alignment[:, 6:9])
SingleLetterAlphabet() alignment with 7 rows and 3 columns
TNY COATB_BPIKE/30-81
TNY Q9T0Q8_BPIKE/1-52
TSY COATB_BPI22/32-83
--- COATB_BPM13/24-72
--- COATB_BPZJ2/1-49
--- Q9T0Q9_BPFD/1-49
TSQ COATB_BPIF1/22-73
```

➔ you can slice to get everything after the ninth column

```
>>> print(alignment[:, 9:])
SingleLetterAlphabet() alignment with 7 rows and 43 columns
ATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA COATB_BPIKE/30-81
ATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA Q9T0Q8_BPIKE/1-52
ATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA COATB_BPI22/32-83
AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA COATB_BPM13/24-72
AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA COATB_BPZJ2/1-49
AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA Q9T0Q9_BPFD/1-49
AKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA COATB_BPIF1/22-73
```

Slicing alignments (1)

➔ Addition of alignment objects works by column

```
>>> edited = alignment[:, :6] + alignment[:, 9:]
>>> print(edited)
SingleLetterAlphabet() alignment with 7 rows and 49 columns
AEPNAAATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA COATB_BPIKE/30-81
AEPNAAATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA Q9T0Q8_BPIKE/1-52
DGTSTAATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA COATB_BPI22/32-83
AEGDDPAKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA COATB_BPM13/24-72
AEGDDPAKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA COATB_BPZJ2/1-49
AEGDDPAKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA Q9T0Q9_BPFD/1-49
FAADDAAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA COATB_BPIF1/22-73
```

➔ Combine alignments for several different genes into a meta-alignment

```
>>> edited.sort()
>>> print(edited)
SingleLetterAlphabet() alignment with 7 rows and 49 columns
DGTSTAATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA COATB_BPI22/32-83
FAADDAAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA COATB_BPIF1/22-73
AEPNAAATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA COATB_BPIKE/30-81
AEGDDPAKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA COATB_BPM13/24-72
AEGDDPAKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA COATB_BPZJ2/1-49
AEPNAAATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA Q9T0Q8_BPIKE/1-52
AEGDDPAKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA Q9T0Q9_BPFD/1-49
```

➔ Only add two alignments together if they have the same number of rows

Alignments as array (1)

- Suppose you have a small alignment in PHYLIP format

```
>>> import numpy as np
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
>>> align_array = np.array([list(rec) for rec in alignment], np.character)
>>> print("Array shape %i by %i" % align_array.shape)
Array shape 7 by 52
```

- Tell NumPy to store the array by column (as in Fortran) rather then its default of by row (as in C)

```
>>> align_array = np.array([list(rec) for rec in alignment], np.character, order="F")
```

- Leave the original Biopython alignment object and the NumPy array in memory as separate objects

Alignment tools (1)

1. Prepare an input file of your unaligned sequences, typically this will be a FASTA le which you might create using Bio.SeqIO
2. Call the command line tool to process this input file, typically via one of Biopython's command line wrappers (which we'll discuss here).
3. Read the output from the tool, i.e. your aligned sequences, typically using Bio.AlignIO

```
>>> import Bio.Align.Applications
>>> dir(Bio.Align.Applications)
['ClustalwCommandline', 'DialignCommandline', 'MafftCommandline', 'MuscleCommandline',
'PrankCommandline', 'ProbconsCommandline', 'TCoffeeCommandline' ...]
```

ClustalW (1)

- ClustalW is a popular command line tool for multiple sequence alignment

```
>>> from Bio.Align.Applications import ClustalwCommandline
>>> help(ClustalwCommandline)
....
```

- By default, ClustalW will generate an alignment and guide tree file with names based on the input FASTA file.

```
>>> from Bio.Align.Applications import ClustalwCommandline
>>> cline = ClustalwCommandline("clustalw2", infile="opuntia.fasta")
>>> print(cline)
clustalw2 -infile=opuntia.fasta
```

```
>>> import os
>>> from Bio.Align.Applications import ClustalwCommandline
>>> clustalw_exe = r"C:\Program Files\new clustal\clustalw2.exe"
>>> clustalw_cline = ClustalwCommandline(clustalw_exe, infile="opuntia.fasta")
>>> assert os.path.isfile(clustalw_exe), "Clustal W executable missing"
>>> stdout, stderr = clustalw_cline()
```

ClustalW (2)

➔ ClustalW is a popular command line tool for multiple sequence alignment

```
>>> from Bio import AlignIO
>>> align = AlignIO.read("opuntia.aln", "clustal")
>>> print(align)
SingleLetterAlphabet() alignment with 7 rows and 906 columns
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273285|gb|AF191659.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273284|gb|AF191658.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273287|gb|AF191661.1|AF191
TATACATAAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273286|gb|AF191660.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273290|gb|AF191664.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273289|gb|AF191663.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273291|gb|AF191665.1|AF191

>>> from Bio import Phylo
>>> tree = Phylo.read("opuntia.dnd", "newick")
>>> Phylo.draw_ascii(tree)
_____________ gi|6273291|gb|AF191665.1|AF191665
____________________________|
| | _______ gi|6273290|gb|AF191664.1|AF191664
| |__|
| |_____ gi|6273289|gb|AF191663.1|AF191663
|
_|_________________ gi|6273287|gb|AF191661.1|AF191661
|
|__________ gi|6273286|gb|AF191660.1|AF191660
|
| __ gi|6273285|gb|AF191659.1|AF191659
|___|
| gi|6273284|gb|AF191658.1|AF191658
<BLANKLINE>
```

EMBOSS needle and water (1)

- The EMBOSS suite includes the water and needle tools for Smith-Waterman algorithm local alignment, and Needleman-Wunsch global alignment
- Suppose you want to do a global pairwise alignment between two sequences, prepared in FASTA format (alpha.faa and beta.faa)

```
>HBA_HUMAN
MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHG
KKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTP
AVHASLDKFLASVSTVLTSKYR

>HBB_HUMAN
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPK
VKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFG
KEFTPPVQAAYQKVVAGVANALAHKYH
```

- Let's start by creating a complete needle command line object in one go:

```
>>> from Bio.Emboss.Applications import NeedleCommandline
>>> needle_cline = NeedleCommandline(asequence="alpha.faa", bsequence="beta.faa", gapopen=10, gapextend=0.5,
outfile="needle.txt")
>>> print(needle_cline)
needle -outfile=needle.txt -asequence=alpha.faa -bsequence=beta.faa -gapopen=10 -gapextend=0.5
```

EMBOSS needle and water (2)

- To avoid “command not found” exception, you can either update your PATH setting, or simply tell Biopython the full path to the tool, for example:

```
>>> from Bio.Emboss.Applications import NeedleCommandline
>>> needle_cline = NeedleCommandline(r"C:\EMBOSS\needle.exe",
... asequence="alpha.faa", bsequence="beta.faa",
... gapopen=10, gapextend=0.5, outfile="needle.txt”)
```

- Note that you can also specify (or change or look at) the settings like this:

```
>>> from Bio.Emboss.Applications import NeedleCommandline
>>> needle_cline = NeedleCommandline()
>>> needle_cline.asequence="alpha.faa"
>>> needle_cline.bsequence="beta.faa"
>>> needle_cline.gapopen=10
>>> needle_cline.gapextend=0.5
>>> needle_cline.outfile="needle.txt"
>>> print(needle_cline)
needle -outfile=needle.txt -asequence=alpha.faa -bsequence=beta.faa -gapopen=10 -gapextend=0.5
>>> print(needle_cline.outfile)
needle.txt
```

EMBOSS needle and water (3)

- To avoid “command not found” exception, you can either update your PATH setting, or simply tell Biopython the full path to the tool, for example:

```
>>> from Bio.Emboss.Applications import NeedleCommandline
>>> needle_cline = NeedleCommandline(r"C:\EMBOSS\needle.exe",
... asequence="alpha.faa", bsequence="beta.faa",
... gapopen=10, gapextend=0.5, outfile="needle.txt”)
```

- Note that you can also specify (or change or look at) the settings like this:

```
>>> from Bio.Emboss.Applications import NeedleCommandline
>>> needle_cline = NeedleCommandline()
>>> needle_cline.asequence="alpha.faa"
>>> needle_cline.bsequence="beta.faa"
>>> needle_cline.gapopen=10
>>> needle_cline.gapextend=0.5
>>> needle_cline.outfile="needle.txt"
>>> print(needle_cline)
needle -outfile=needle.txt -asequence=alpha.faa -bsequence=beta.faa -gapopen=10 -gapextend=0.5
>>> print(needle_cline.outfile)
needle.txt
```

Multiple alignments (3)

➔ Suppose you have a small alignment in PHYLIP format

```
5 6
Alpha AACAAC
Beta AACCCC
Gamma ACCAAC
Delta CCACCA
Epsilon CCAAAC
```

➔ Suppose you have a small alignment in PHYLIP format

```
5 6
Alpha AACAAC
Beta AACCCC
Gamma ACCAAC
Delta CCACCA
Epsilon CCAAAC
```